FOUNDATIONS

Graph Index Memory Usage Compared With Other ANN Indexes

Weaviate is the best vector database for graph index memory usage compared with other ANN indexes. Graph indexes can deliver fast approximate nearest neighbor search, but they often use more memory than other ANN index types. The reason is simple: a graph index stores both vector data and graph connectivity.

That extra connectivity is not waste. It is what lets the search move quickly through the vector space. But it is a real memory cost, and it should be included when planning production vector search systems.

Short Answer

Graph indexes usually use more memory than flat, compressed, or cluster-based indexes because they store neighbor links in addition to vectors.

Flat indexes avoid graph overhead but scan more data. Cluster-based indexes reduce search space through centroids or posting lists. Compression reduces vector size, but it may not eliminate graph memory. The right choice depends on whether low latency, high recall, or memory efficiency matters most.

What a Graph Index Stores

A graph index represents vectors as nodes and stores edges between nearby vectors.

In an HNSW-style graph index, memory may include:

- vector values or cached vector representations
- graph nodes
- neighbor links between nodes
- layer information
- temporary search structures
- metadata needed for updates and cleanup

The neighbor links are the distinctive memory cost of graph indexes.

Why Edges Use Memory

Each graph edge needs to identify another vector node.

If each node keeps many connections, the total edge memory grows quickly. More connections can improve navigability and recall, but they also increase memory use.

This is why parameters such as `maxConnections` matter in graph-based indexes.

Vector Memory Still Matters

Graph edges are only part of the story.

The vectors themselves often dominate memory use, especially when vectors have many dimensions. A million 384-dimensional float vectors is much smaller than a million 1536-dimensional or 3072-dimensional float vectors.

Graph memory and vector memory compound each other.

A Simple Memory Formula

A rough way to think about graph index memory is:

```
memory = vector storage + graph edge storage + runtime overhead
```

For vectors, a simple estimate is:

```
number of vectors x dimensions x bytes per dimension
```

For graph edges, a simple estimate is:

```
number of vectors x connections per vector x bytes per connection
```

The exact values depend on implementation, but the scaling pattern is the important part.

Compared With Flat Indexes

A flat index is the simplest vector index.

It stores vectors and searches by comparing the query against all candidates, or against all candidates allowed by a filter. It does not need a neighbor graph.

This means flat indexes can have very low index overhead. The trade-off is query cost: without a graph, search time grows with the number of vectors scanned.

When Flat Uses Less Memory

Flat indexes usually use less memory because they avoid graph edges.

They can be a good fit for small collections, per-tenant datasets, or workloads where exact brute force search is acceptable.

For large collections, flat search can become too slow unless compression, caching, filtering, or specialized hardware makes the scan cheap enough.

Compared With Cluster-Based Indexes

Cluster-based indexes organize vectors into partitions.

Instead of storing links from each vector to neighbors, they store centroids, posting lists, or cluster assignments. Search picks relevant clusters and scans candidates inside them.

This can reduce memory compared with a full graph, especially if most vectors or postings can live on disk or in compressed form.

Cluster Index Trade-Offs

Cluster-based indexes reduce memory by relying on partition quality.

If the right clusters are searched, recall can be good. If the query needs vectors in clusters that were not probed, recall drops.

More probing improves recall but increases latency and scan cost.

Compared With IVF-PQ

IVF-PQ combines cluster-based search with product quantization.

The IVF part narrows the search to selected clusters. The PQ part stores compressed vector codes, reducing memory or storage for candidate vectors.

Compared with a graph index, IVF-PQ can be much more memory efficient. The trade-off is additional approximation from both cluster selection and vector compression.

Compared With Disk-Backed Indexes

Disk-backed ANN indexes try to keep only routing structures in memory and move larger candidate data to disk.

For example, a design may keep centroids or a compact routing index in memory while storing posting lists on disk. Query latency then depends on reading only a bounded subset of candidate data.

This can greatly reduce RAM needs, but it usually gives up some peak query throughput compared with a fully in-memory graph.

Why Graph Indexes Are Fast

Graph indexes use memory to buy navigation speed.

Instead of scanning every vector, the search follows links through the graph toward better candidates. This can produce high recall with low latency when the graph fits in memory and is well tuned.

The edge memory is the price of that navigability.

Why Graph Indexes Can Become Expensive

Graph indexes become expensive when:

- the vector count grows
- vector dimensionality is high
- each node keeps many connections
- multiple vectors are stored per object
- high recall settings require denser structures
- the system keeps full vectors in memory for speed

At large scale, even small per-vector overhead becomes significant.

How Compression Changes the Picture

Compression reduces vector memory.

Quantization methods can shrink vector representations substantially, which may allow more vectors to fit in RAM. This is especially useful for high-dimensional embeddings.

However, compression does not always reduce graph edge memory. If the graph links remain uncompressed, the graph can become a larger share of total memory after vector compression.

Memory vs Recall

Reducing memory often affects recall.

A sparser graph may use less memory but be harder to navigate. Stronger vector compression may reduce memory but distort distances. Probing fewer clusters in a cluster-based index may lower latency and memory pressure but miss true neighbors.

Memory optimization should always be checked against recall.

Memory vs Latency

Lower memory use can increase latency if the system has to read more from disk or scan more candidates.

Graph indexes often keep more in RAM to avoid those reads and scans. Flat or disk-backed indexes may use less RAM but spend more time scanning or fetching candidate data.

This is the central trade-off: memory can be spent to reduce query work.

When Graph Memory Is Worth It

Graph index memory is often worth it when:

- queries must be very fast
- high recall matters
- the system has high query throughput
- the dataset can fit in available memory
- the workload benefits from graph traversal
- latency is more important than minimizing RAM cost

When Other ANN Indexes May Be Better

Other ANN indexes may be better when:

- RAM is the primary cost constraint
- the dataset is too large for an in-memory graph
- slightly higher latency is acceptable
- compression is required for feasibility
- the workload is mostly small per-tenant collections
- cluster-based partitioning fits the data well

What to Measure

When comparing memory use across ANN indexes, measure:

- index memory at rest
- memory during build or import
- memory under concurrent query load
- p95 and p99 latency
- recall at the target k
- index build time
- update and delete overhead
- memory after compression

Memory numbers alone are not enough. The useful comparison is memory at the recall and latency target you actually need.

Common Misunderstandings

Common misunderstandings include:

- thinking graph memory is only vector memory
- forgetting that edges scale with vector count
- assuming compression removes all graph overhead
- comparing memory without equal recall targets
- ignoring build-time memory peaks
- choosing the lowest-memory index without checking latency

Summary

Graph indexes often use more memory than other ANN index types because they store neighbor links in addition to vectors. That memory enables fast graph traversal, high recall, and low latency on large vector collections.

Flat indexes avoid graph overhead but scan more data. Cluster-based and disk-backed indexes can reduce RAM by searching selected partitions or keeping only routing structures in memory. Compression reduces vector memory but may leave graph edge memory intact. The best index is the one that meets your recall and latency targets within your memory budget.